Shastri 6th Semester
Unit-4
Introduction to Data Structures

Systems Programming

Intro to Data Structures

- Self-referential Structures
- Dynamic Memory Allocation
- A Simple malloc Example
- Linear Lists
- Linked Lists
- Insertion Example
- Linked List Example

Intro to Data Structures

- **Data Structure Definition type operation and classification**
- Importance of Data in Data Structure: आज की तारीख मे इस मॉडर्न techno दुनिया मे Data एवं information हमारी लाइफ का बहुत जरुरी element हो गया है एवं इसको अच्छी तरह से organize करते हुए रखने के लिए बहुत अलग अलग तरह के implementation किये गए हैं| Data तो केवल fact and figures का collection है ya आप कह सकते है की डाटा तो केवल एक format मे values या set of values है| एक डाटा item single set of values को कहते है| उसके बाद डाटा items को आगे कई sub items मे categorized किया जा सकता है जैसे की एक student का नाम तीन sub items मे divide किया जा सकता है -first name, middle name and last name लेकिन student को जो ID assign होगा उसकी केवल एक ही value होगी और उसको single item consider किया जायेगा |

Intro to Data Structures

- डाटा स्ट्रक्चर क्या है : Data Structure, डाटा को systematic तरीके से organize व् collect करने का एक तरीका है जिसमे की किसी भी तरह का operation (जैसे insertion, deletion) easily एवं effectively perform किया जा सके| Data को अच्छी तरह से organize करने से memory space को अच्छी तरह से utilize एवं time की बचत भी की जा सकती है जिससे अलग अलग task की complexities दूर होती है| यह इस तरह designed and implemented किया होना चाहिए की यह complexity को कम करते हुए efficiency को बढ़ाये| computer Data structure की सबसे छोटी or simple definition है की जो भी data को systematic way से store कर सकता है उसे डाटा स्ट्रक्चर कहते है|
- Basic Operations : डाटा स्ट्रक्चर के data पर बहुत सारे अलग अलग operations perform किये जाते है | Data पर किये जा सकने वाले कुछ important operations है –
- Searching
- Insertion
- Deletion
- Sorting
- Traversing
- Merging

Intro to Data Structures

- Categories of Data Structure : Data structure को दो major category मे divide किया गया है -
- Primitive
- Non-primitive data structures
- Primitive Data Structures (Built in data type): Primitive Data Structures basic data structures होते है जो की directly machine instructions पर operate होते है। ये अलग अलग computers पर अलग अलग तरह से represent होते है। Integer, Float, character constants, string constants and pointers इस category मे आते है। इनको built-in data टाइप्स भी कहते है क्योकि ये वो डाटा टाइप है जिस पर language का built-in support होता है।
- Non-primitive Data Structures (Derived data type) : Non-primitive data structures जयदा complicated data structures होते है एवं and इनको primitive data structures से ही लिया गया है। Array, Lists, Files, Stacks, Queues, Graphs and trees इस

Intro to Data Structures

- category मे आते है । इनको derived data टाइप्स भी कहते है क्योकि यह independently बिल्ट इन डाटा टाइप को use करते हुए इम्प्लीमेंट (derive) किये जा सकते है। यह data types, built-in data types और उनसे रिलेटेड ऑपरेशन्स के कॉम्बिनेशन से बनते है|
- Non-primitive data structure को दो main category मे divide किया जा सकता है -
- Linear Data Structure
- Non-linear Data Structure
- Linear Data Structure : किसी भी डाटा स्ट्रक्चर को linear कहा जायेगा अगर इसके elements एक specific order को follow करते हुए combine होते है| Examples of linear data structures are –
- Arrays
- Queues
- Stacks
- Linked lists
- Non linear Data Structure : यह डाटा स्ट्रक्चर ज्यादातर उस data को represent करते है जहाँ पर अलग अलग elements के बिच hierarchical relationship होती है | Examples of Non Linear Data Structures –
- Graphs
- family of trees and
- table of contents

Self-Referential Structures

- Self-referential structures contain a pointer member that points to a structure of the same structure type.

Example:

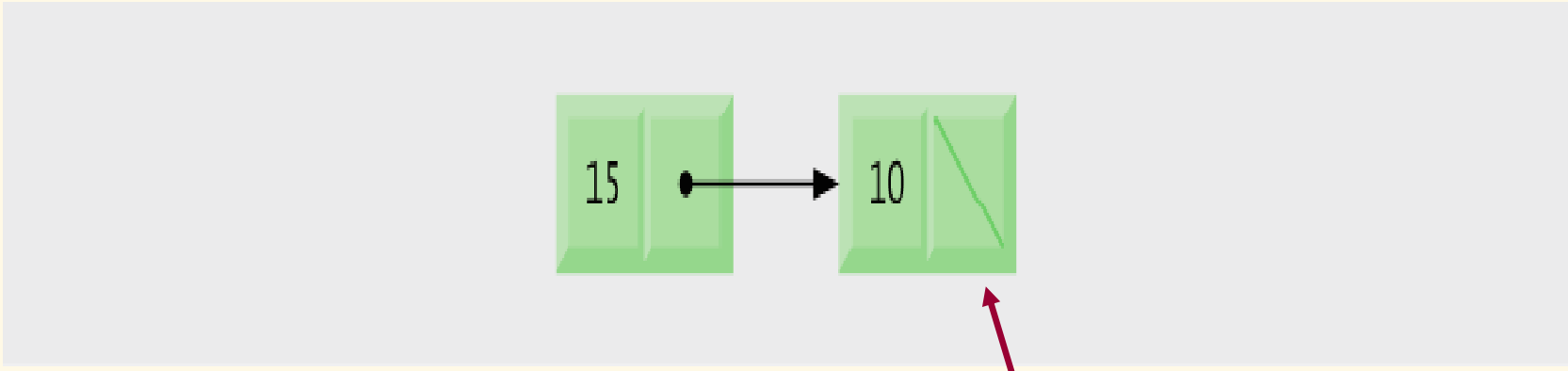
```
struct node {
   int data;
   struct node *nextPtr;
}
```

- nextPtr
  - is a pointer member that points to a structure of the same type as the one being declared.
  - is referred to as a link. Links can tie one node to another node.
- Self-referential structures can be linked together to form useful data structures such as lists, queues, stacks and trees.

Fig. 12.1 Self-referential structures linked together

- Structure ये एक अलग-अलग data types का collection होता है |
- अगर structure का इस्तेमाल करना हो तो 'struct' keyword का इस्तेमाल करते है |
- Structure ये array के जैसा ही होता है |
- Array similar data types का collection होता है और Structure different data type का collection होता है |
- Structure में किये हुए हर variable के decaration को 'member' कहते है |
- Structure हर एक member के लिए अलग-अलग memory allocate करता है |

Fig. 12.1 Self-referential structures linked together



Not setting the link in the last node of a list to NULL can lead to runtime errors.

12.3 Dynamic Memory Allocation

- Creating and maintaining dynamic data structures requires dynamic memory allocation, namely, the ability obtain and release memory at execution time.
- In C, the functions malloc and free and the operator sizeof are essential to dynamic memory allocation.

Dynamic Memory Allocation

- **Dynamic Memory Allocation:** Computer में कोई भी Program इसलिए बनाया जाता है ताकि विभिन्न प्रकार के Data को Process किया जा सके और किसी समस्या का समाधान प्राप्त किया जा सके। विभिन्न प्रकार के Data को Memory में Store करने के लिए हम विभिन्न प्रकार के Variables Create करते हैं।
- वास्तविक जीवन में हमेंशा ये निश्चित नहीं होता कि कितने Data के साथ प्रक्रिया करनी है। यानी मानलो कि किसी Company में 20 Employee काम करते हैं। उस Company में आवश्यकतानुसार किसी Employee को Company से निकाला भी जा सकता है और किसी नए Employee को Company में Appoint भी किया जा सकता है।
- यानी ये निश्चित नहीं होता कि हमेंशा उस Company में 20 Employee ही काम करेंगे। इसलिए यदि किसी Company के Employees का Record रखने के लिए यदि Computer में कोई Program Develop किया जा रहा है तो Computer में भी हम एक निश्चित संख्या में Variables Declare नहीं कर सकते।

Dynamic Memory Allocation

- Computer में भी हमें एक ऐसी व्यवस्था की जरूरत होगी जिससे यदि Data बढते हैं तो नए Variables Create हो सकें और यदि Data घटते हैं तो किसी पुराने Variable को Delete किया जा सके ताकि उस Variable द्वारा Reserve की गई Space का कोई अन्य Program उपयोग कर सके।
- "C" Language में इस काम को करने के लिए Dynamic Memory Allocation Concept को Use किया जाता है। यानी हम Program में अपनी आवश्यकतानुसार Variables Create कर सकते हैं और उन्हें मान प्रदान कर सकते हैं। इस व्यवस्था को **Dynamic Memory Allocation** कहा जाता है।
- Dynamic Memory Allocation के लिए "C" में कुछ Memory Management Functions हैं जिनका प्रयोग करके हम Program के Run Time में विभिन्न प्रकार के Variables Create करके उन्हें Memory प्रदान कर सकते हैं। ये Function Memory में Data के लिए Storage Space बनाने या Space हटाने का काम करते हैं।

A Simple malloc Example

malloc() (Memory Allocation): Standard Library Function

malloc() ये Memory का block allocate करता है |

Syntax for malloc()

```
pointer = (cast_type*)malloc(size_of_block);
```

calloc() ये function एक Dynamic Memory Allocation का प्रकार है |

malloc() function सिर्फ एक Memory block allocate करता है |

अगर ये memory allocate नहीं करता तो वो NULL return करता है |

malloc() function में जब Memory allocate होती है, तब इसकी block पर garbage value होती है |

malloc

- malloc
  - Takes as its argument the number of bytes to allocate.
    - Thus, sizeof is used to determine the size of an object.
  - Returns a pointer of type void *
    - A void * pointer may be assigned to any pointer.
    - If no memory is available, malloc returns NULL.

Example

```
newPtr = malloc( sizeof( struct node ))
```

Note – the allocated memory is NOT initialized.

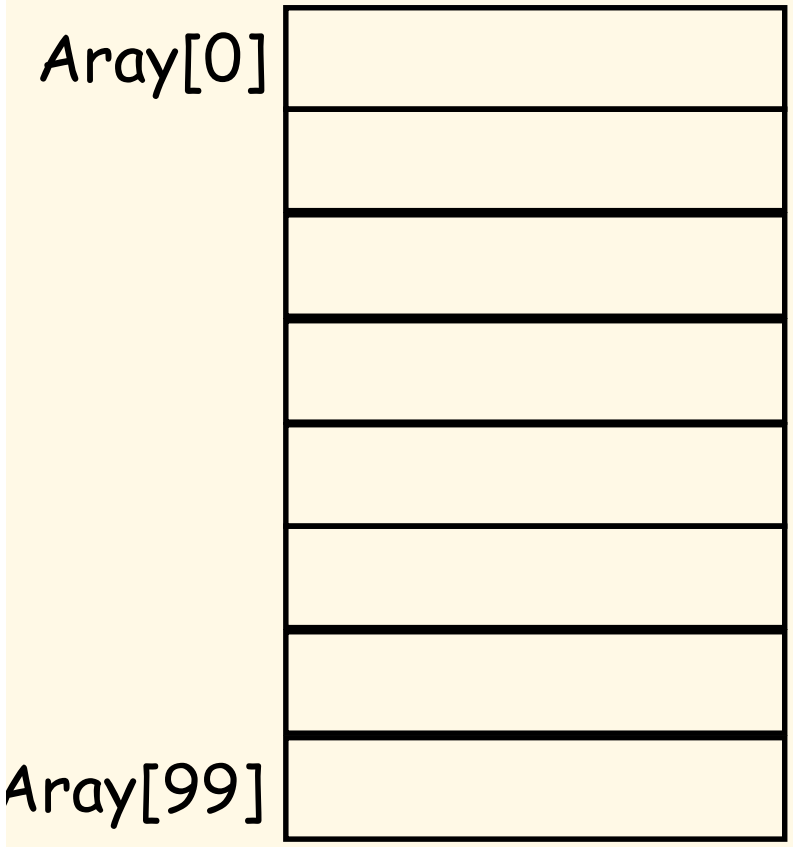
free

- free
  - Deallocates memory allocated by malloc.
  - Takes a pointer as an argument.

Example:

```
free ( newPtr );
```

Linear Lists

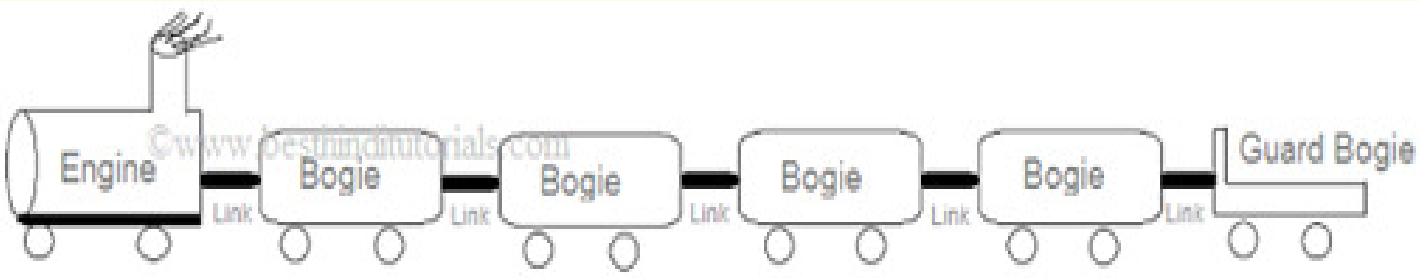


- With a linear list, the assumption is the next element in the data structure is implicit in the index.
- This saves space, but is expensive for insertions!

Introduction to Linked List

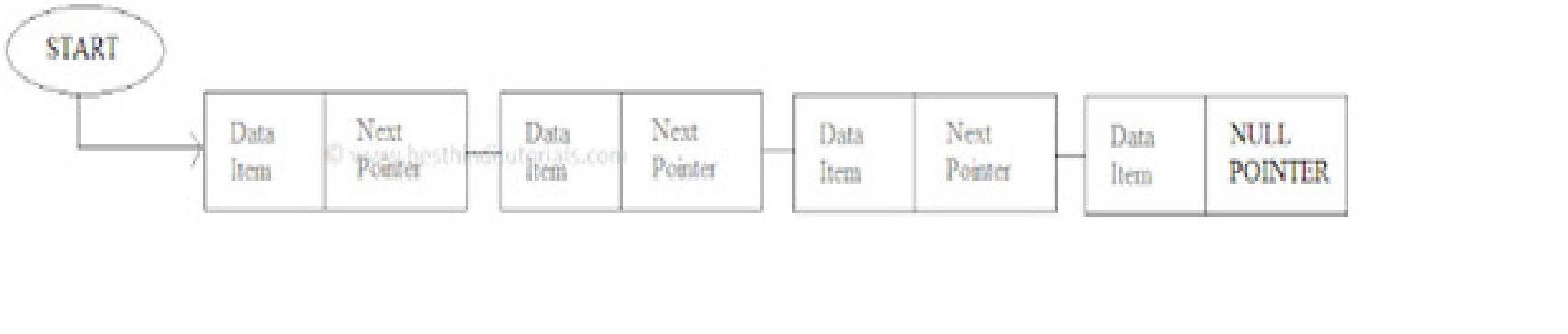
एक linked list linear (sequential) data structure होती है। जब आप कोई list बनाते है तो उसमे एक के बाद एक item लिखते जाते है। उसी प्रकार linked list में भी एक data item के बाद दूसरा item add किया जाता है।

Linked list को आप किसी train की तरह समझ सकते है। एक train की सभी bogies आपस में connected होती है। आप एक bogie से दूसरी bogie में जा सकते है। उसी प्रकार एक linked list के सभी data items आपस connected होते है।



Introduction to Linked List

Linked list में add किया गया हर item उससे next item के address को hold करता है और memory में point करता है की next item कँहा पर stored है। इससे अलग अलग locations पर data items stored होने पर भी linked list ऐसे behave करती है जैसे सभी data items एक बाद एक contagious memory locations में store किये गए हो।



Introduction to Linked List

- पहला variable data को hold करता है। दूसरा variable define किये जा रहे structure का ही pointer variable होता है। Pointer variable को next या link जैसे नाम दिए जाते है। यह variable next node या data item के address को store करता है।
- **Array vs Linked List**
- सामन्यतः linked list array की ही तरह प्रतीत होती है। लेकिन कुछ ऐसी असमानताएँ है जो इन दोनों को अलग बनाती है। आइये इन असमानताओं के बारे में जानने का प्रयास करते है।
- एक linked list की size array की तरह fix नहीं होती है। जब आप array create करते है तो साथ ही उसकी size भी define करते है। आप define की गयी size से अधिक elements array में add नहीं कर सकते है। लेकिन linked list की size fix नहीं होती है और आप कितने भी element linked list में add कर सकते है।
- एक linked list में data dynamically store किया जा सकता है। लेकिन array में यह कर पाना असम्भव है।

Introduction to Linked List

- एक linked list में आप pointers को re arrange करके data items को आसानी से insert और delete कर सकते है। लेकिन array में insertion और deletion perform करने के लिए data items के बहुत अधिक movement की आवश्यकता होती है।
- एक linked की size को dynamically grow और shrink किया जा सकता है। Array के साथ ऐसा कर पाना संभव नहीं है।
- एक linked list array से अधिक memory का प्रयोग करती है। इसका कारण यह है की linked list में data items के साथ next pointers भी attached होते है।
- एक array के elements को index number द्वारा randomly access किया जा सकता है। लेकिन linked list के elements को आप randomly access नहीं कर सकते है।
- एक linked list के elements को traverse करने में array से अधिक समय लगता है। इसका कारण extra next pointer होता है।
- Linked list data structure में binary searching नहीं perform किया जा सकता है।
- एक array के सभी elements continuous memory locations में store किये जाते है। लेकिन linked list के elements अलग अलग memory locations पर stored हो सकते है। Linked list द्वारा memory का सही utilization किया जाता है।

Introduction to Linked List

- Types of Linked List
- Linked list तीन प्रकार की होती है। इनके बारे में निचे बताया जा रहा है।
- **Singly Linked List**
- एक singly linked list सबसे सरल और सामान्य linked list होती है। Singly linked list का हर item उससे अगले item का address store करता है। इस तरह की linked list में सिर्फ एक ही address store किया जाता है। यह भी एक कारण है की इसे singly linked list कहा जाता है।
- **Doubly Linked list**
- एक doubly linked list singly linked list की तरह ही होती है। लेकिन doubly linked list में next pointer के अलावा previous pointer भी होता है। Next pointer जँहा अगले element का address store करता है वँही previous pointer previous element का address store करता है।
- Doubly linked list में reverse traversal किया जाना संभव होता है।
- **Circular Linked List**

Introduction to Linked List

- Circular linked list singly linked list ही होती है। लेकिन इस तरह की linked list में आखिर element NULL की बजाय first element का address store करता है। इस प्रकार circular linked list एक loop या cycle की तरह होती है।
- Operations with Linked List
- एक Linked list के साथ निचे दिए जा रहे operations perform किये जा सकते है।
- Creating a linked list
- Traversing or Displaying a linked list
- Counting nodes of linked list
- Inserting a linked list
- Deleting a linked list
- Searching a linked list

12.4 Linked Lists

- Linked list
  - A linear collection of self-referential class objects, called nodes.
  - Connected by pointer links.
  - Accessed via a pointer to the first node of the list.
  - Subsequent nodes are accessed via the link-pointer member of the current node.
  - The link pointer in the last node is set to NULL to mark the list's end.
- Use a linked list instead of an array when
  - You have an unpredictable number of data elements.
  - Your list needs to be sorted quickly.

Fig. 12.2 Linked list graphical representation

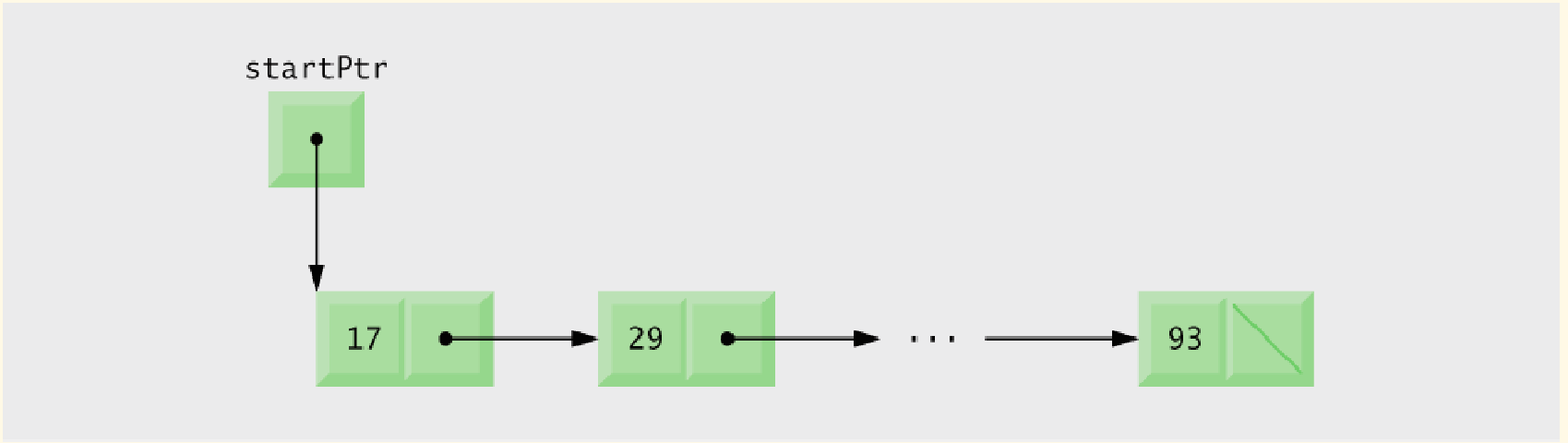


Fig. 12.5 Inserting a node in an ordered list

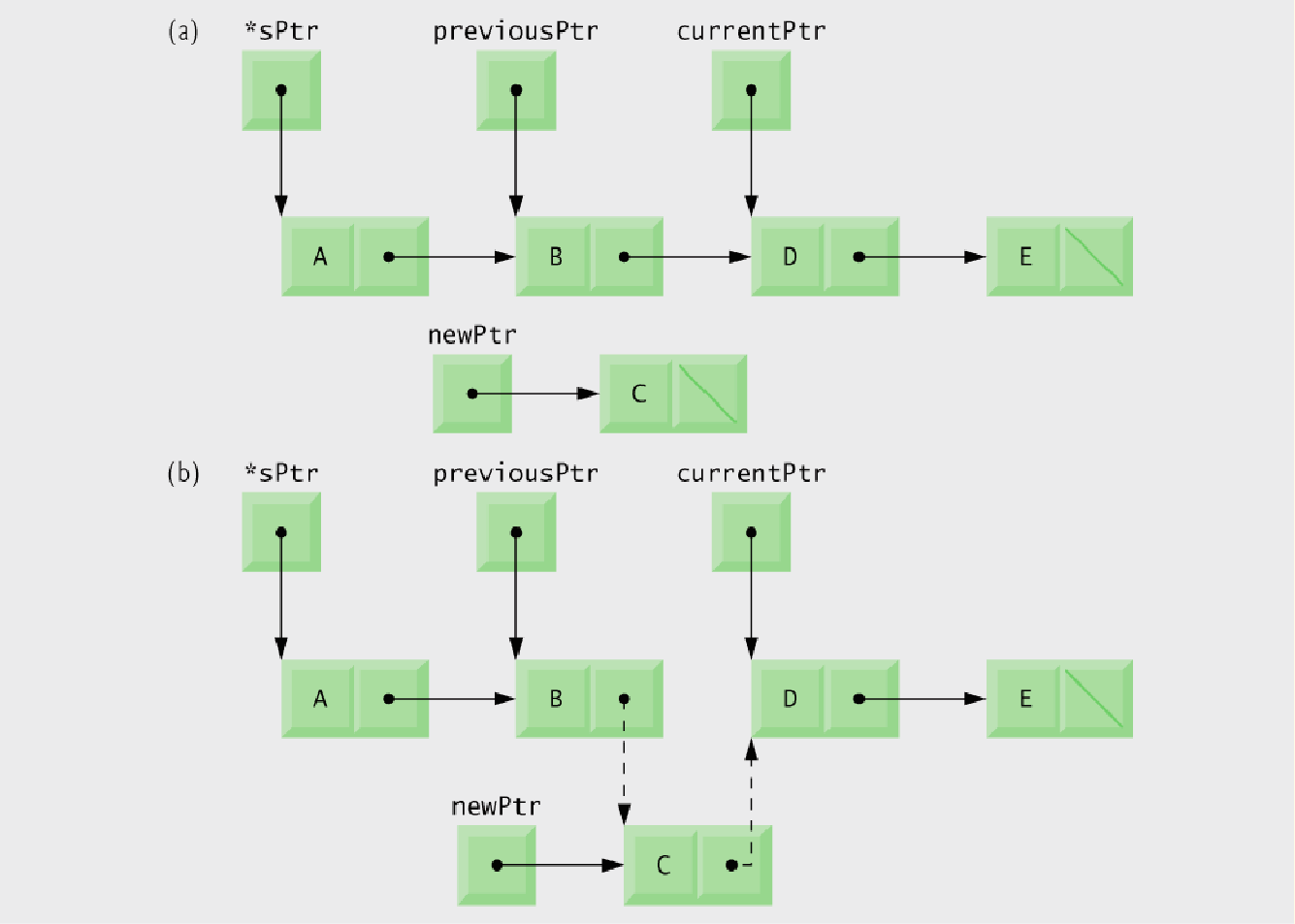
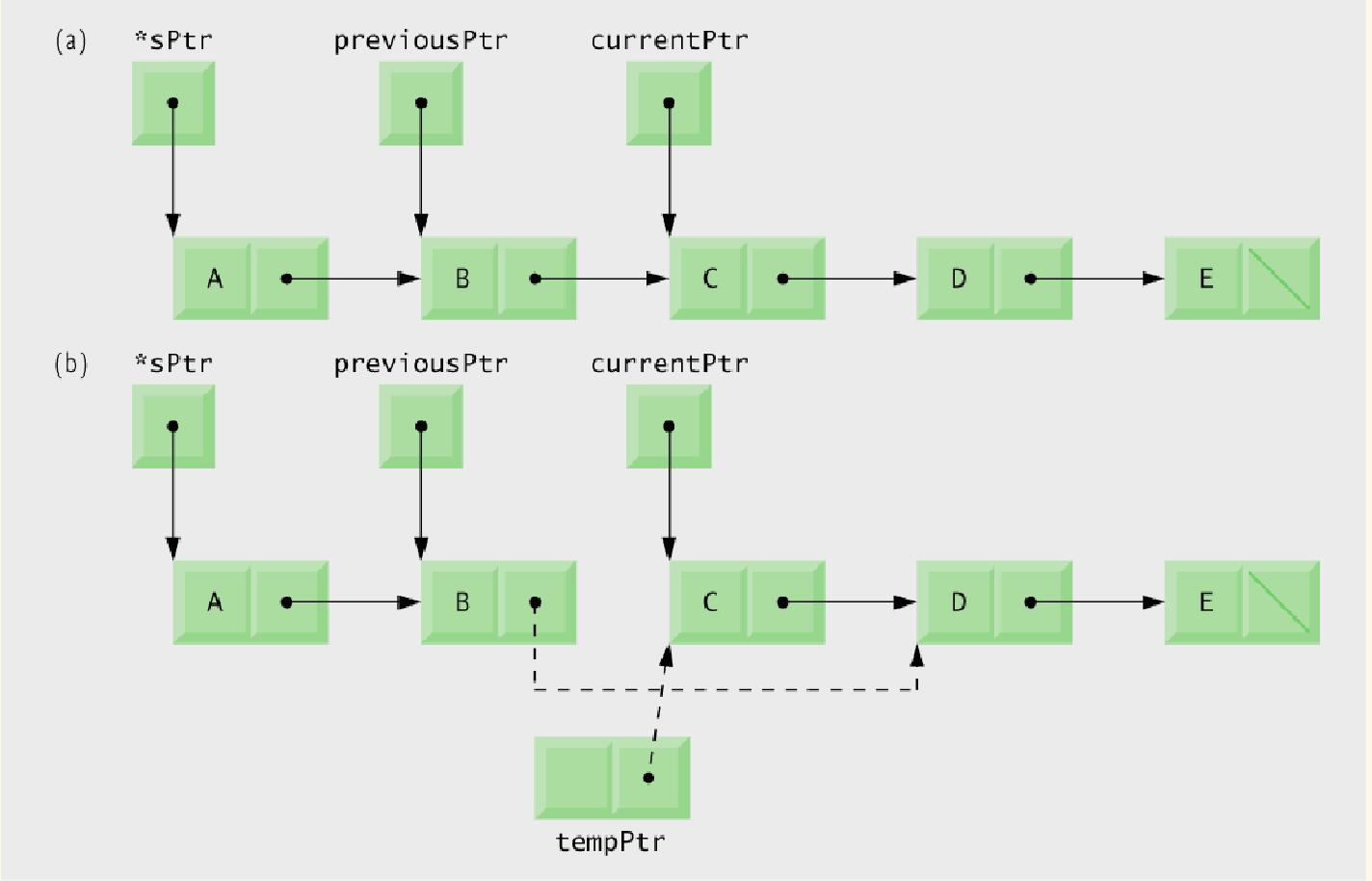
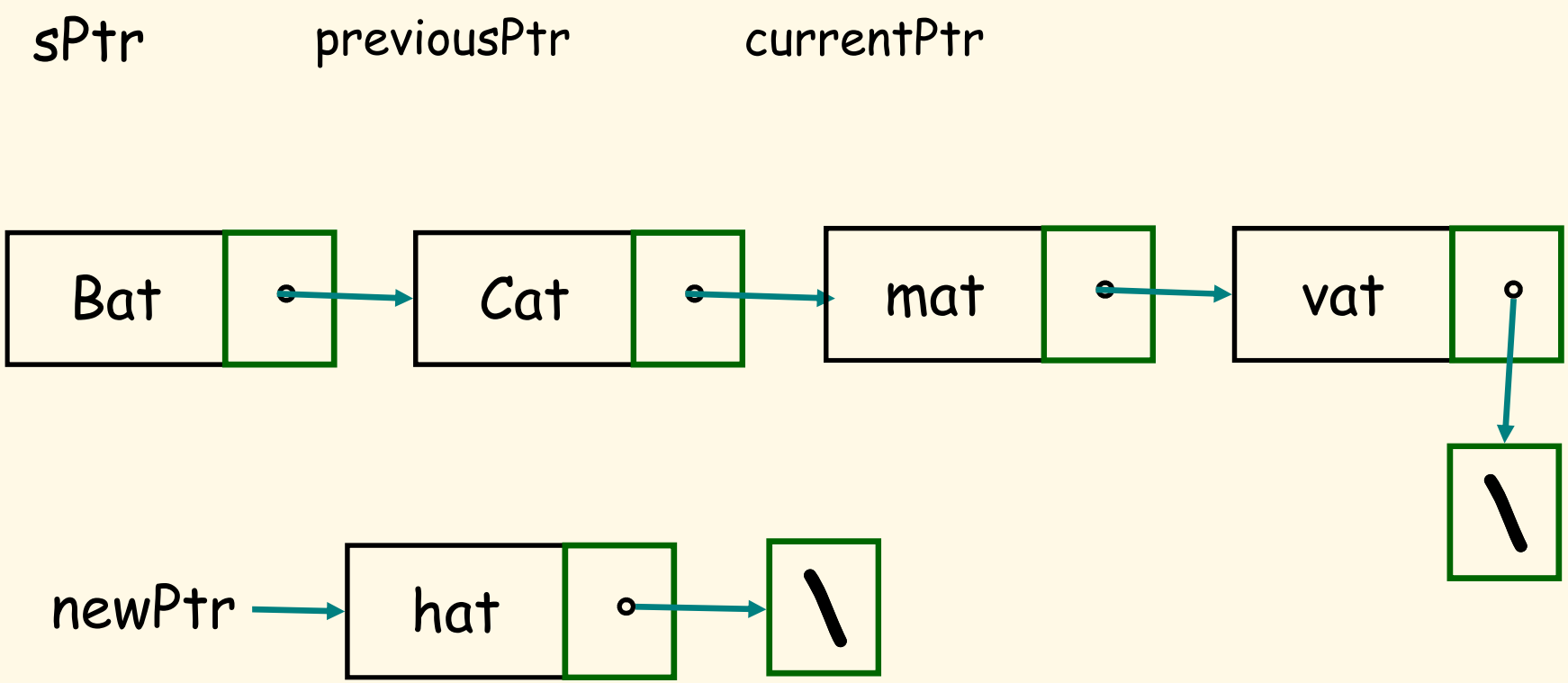


Fig. 12.5 Deleting a node in an ordered list



Linked List Insertion Example



Summary/Review

- Introduced malloc and free.
- Discussed the tradeoffs between a linear list and a linked list.
- Provided two linked examples.
- Explained event lists as one instance of linked lists.
- Important operations on linked lists:
  - Insertion in the list.
  - Taking a random node out of the list.
  - Taking the 'next' node off the front of the list.
  - Printing the linked list in order

Review of Data Structures Seen

- Arrays
- Arrays of pointers to strings
- Arrays of structs (note: this is a linear list)
- Linked lists
  - singly-linked lists
  - [not] doubly-linked lists!!